Driver Type Clasification(Supervised)-FCD Output with other relevant feature

In [1]:
```
import pandas as pd
```

In [2]:
```
fcd_df = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/fcd-output
emission_df = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/emiss
features = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/features
```

In [3]:
```
features
```

Out[3]:

	Unnamed: 0	time	id	x	y	angle	type	speed	pos	lane
0	0	0.0	f_0.0	-38.09	-4.80	90.00	SportsCarDriver	0.00	61.91	E0_0
1	1	0.1	f_0.0	-38.08	-4.80	90.00	SportsCarDriver	0.03	61.92	E0_0
2	2	0.2	f_0.0	-38.07	-4.80	90.00	SportsCarDriver	0.10	61.93	E0_0
3	3	0.3	f_0.0	-38.05	-4.80	90.00	SportsCarDriver	0.21	61.95	E0_0
4	4	0.4	f_0.0	-38.01	-4.80	90.00	SportsCarDriver	0.40	61.99	E0_0
...	...	...	...	...	...	...	...	...	...	...
398475	398475	444.3	f_5.99	87.30	10.02	294.06	CautiousDriver	16.62	5.14	:J2_0_0
398476	398476	444.4	f_5.99	85.91	10.87	300.53	CautiousDriver	16.55	6.79	:J2_0_0
398477	398477	444.5	f_5.99	84.29	11.17	296.05	CautiousDriver	16.48	8.44	:J2_0_0
398478	398478	444.6	f_5.99	82.66	11.27	285.32	CautiousDriver	16.41	1.02	E8_0
398479	398479	444.7	f_5.99	81.03	11.25	274.53	CautiousDriver	16.34	2.65	E8_0

398480 rows × 16 columns

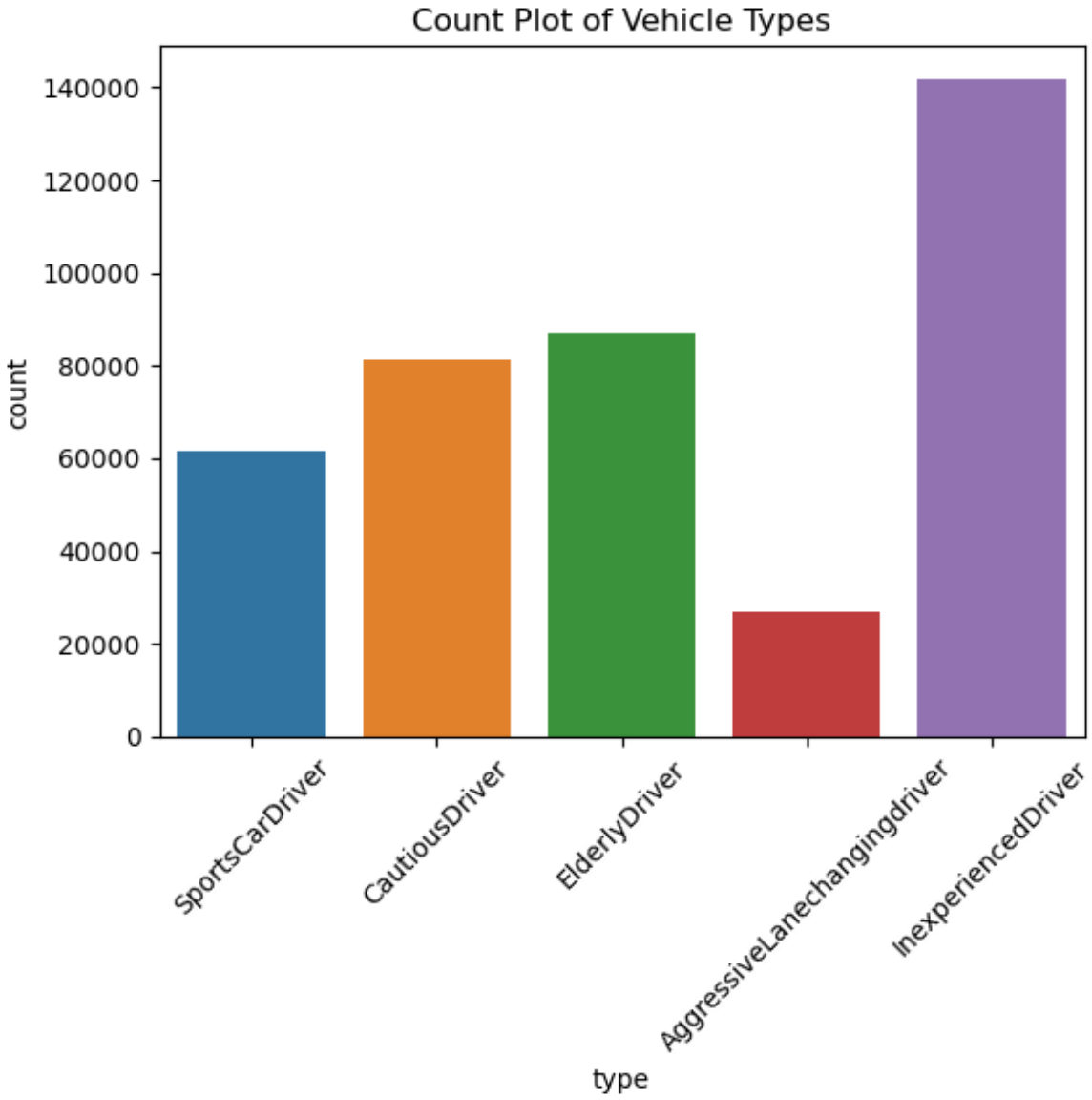
In [4]:
```
# Merging the data using 'id' and 'type'
merged_data = pd.merge(fcd_df,emission_df, on=['time','id','type'])
```

In [5]:
```
# Merging the data using 'id' and 'type'
merged_data = pd.merge(merged_data,features, on=['time','id','type'])
```

Count Plot of Vehicle Types

In [29]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Example of creating a count plot for the 'type' column
sns.countplot(data=merged_data, x='type')
plt.title('Count Plot of Vehicle Types')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if nee
plt.show()
```



In [12]:
```
final = merged_data.drop(columns=['slope','eclass','route','waiting','lane_y'
```

In [13]:
```
final
```

Out[13]:

	time	id	x_x	y_x	angle_x	type	speed_x	pos_x	a
0	0.0	f_0.0	-38.09	-4.80	90.00	SportsCarDriver	0.00	61.91	
1	0.0	f_1.0	-57.03	-37.67	33.37	CautiousDriver	0.00	75.68	
2	0.0	f_2.0	-33.33	-1.60	90.00	ElderlyDriver	0.00	66.67	
3	0.0	f_3.0	357.14	4.80	270.00	AggressiveLanechangingdriver	0.00	42.86	
4	0.0	f_4.0	241.15	21.32	215.86	AggressiveLanechangingdriver	0.00	98.24	
...	...	...	...	...	...	...	...	...	
398475	999.9	f_5.222	111.69	1.60	270.00	SportsCarDriver	0.00	102.38	
398476	999.9	f_5.223	245.35	4.80	270.00	InexperiencedDriver	12.57	55.00	
398477	999.9	f_5.224	158.14	4.80	270.00	SportsCarDriver	26.32	55.93	
398478	999.9	f_5.225	335.19	1.60	270.00	ElderlyDriver	8.05	64.81	
398479	999.9	f_5.226	367.06	4.80	270.00	ElderlyDriver	0.04	32.94	

398480 rows × 14 columns

In [14]:
```
from sklearn.preprocessing import LabelEncoder
# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical columns
final['type'] = label_encoder.fit_transform(final['type'])
final['id'] = label_encoder.fit_transform(final['id'])
```

In [15]:
```
final.corr()
```

Out[15]:

	time	id	x_x	y_x	angle_x	type	sp
time	1.000000	-0.027918	-0.011356	0.005042	0.014711	-0.014541	-0.0
id	-0.027918	1.000000	0.327327	0.409731	0.838318	0.065246	-0.2
x_x	-0.011356	0.327327	1.000000	0.386571	0.391870	0.017162	0.0
y_x	0.005042	0.409731	0.386571	1.000000	0.518102	0.037787	0.0
angle_x	0.014711	0.838318	0.391870	0.518102	1.000000	0.061652	-0.0
type	-0.014541	0.065246	0.017162	0.037787	0.061652	1.000000	0.0
speed_x	-0.047372	-0.203942	0.076041	0.001827	-0.078379	0.039832	1.0
pos_x	0.032194	0.066977	-0.114310	0.007374	-0.005626	0.013670	-0.1
acceleration_x	-0.021207	-0.047929	-0.026528	-0.005661	-0.045198	0.006287	-0.1
angle	0.014711	0.838318	0.391870	0.518102	1.000000	0.061652	-0.0
lane_length	0.015000	0.036952	-0.226934	-0.112893	-0.090701	0.000577	-0.1
lane_change	-0.003851	-0.017214	0.012718	0.002876	0.000320	0.004922	0.0
vehicle_density	-0.004640	0.017209	0.014581	-0.016119	0.016353	-0.004523	-0.0
avg_speed_nearby_vehicles	-0.050949	-0.214982	0.067308	-0.013411	-0.087862	0.029756	0.9

Random Forest(ensemble Method)

In [16]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Assuming 'df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = final.drop(['type','id'],axis=1)  # Features
y = final['type']  # Target variable

# Split the data into training and testing sets (adjust test_size and random_s
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

# Initialize the classifier (you can use any other classifier of your choice)
clf_rand = RandomForestClassifier(n_estimators=200, random_state=42)

# Fit the classifier on the training data
clf_rand.fit(X_train, y_train)

# Predict on the test data
y_pred = clf_rand.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.9686182493475206
```

KNN-Model

In [17]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score



# Features (X) and target variable (y)
X = final.drop(['type','id'], axis=1)  # Features
y = final['type']  # Target variable


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the k-NN classifier
k = 5  # You can change this value as needed
knn = KNeighborsClassifier(n_neighbors=k)

# Fit the model on the training data
knn.fit(X_train, y_train)

# Predict the labels for the test set
y_pred = knn.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the k-NN model: {accuracy:.2f}')
```

```
Accuracy of the k-NN model: 0.90
```

DNN Model

In [18]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras import regularizers
from tensorflow.keras.utils import to_categorical

# Assuming 'fcd_df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = final.drop(['type','id'], axis=1)  # Features
y = final['type']  # Target variable

# Convert y to categorical (one-hot encoded)
y = to_categorical(y)
num_classes = 5
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the Deep Neural Network model with L2 regularization and dropout
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu', kernel_reg
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.0
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.0
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))  # For multi-class classi

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac

# Fit the model on the training data
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test

# Evaluate the model on the test data
_, accuracy = model.evaluate(X_test, y_test)
print(f"Accuracy: {accuracy}")
```

```
Epoch 1/10
9962/9962 [==============================] - 13s 1ms/step - loss: 1.3961 - a
ccuracy: 0.4293 - val_loss: 1.3055 - val_accuracy: 0.4464
Epoch 2/10
9962/9962 [==============================] - 12s 1ms/step - loss: 1.3203 - a
ccuracy: 0.4396 - val_loss: 1.2952 - val_accuracy: 0.4450
Epoch 3/10
9962/9962 [==============================] - 13s 1ms/step - loss: 1.3132 - a
ccuracy: 0.4405 - val_loss: 1.2929 - val_accuracy: 0.4472
Epoch 4/10
9962/9962 [==============================] - 12s 1ms/step - loss: 1.3099 - a
ccuracy: 0.4412 - val_loss: 1.2920 - val_accuracy: 0.4455
Epoch 5/10
9962/9962 [==============================] - 12s 1ms/step - loss: 1.3077 - a
ccuracy: 0.4411 - val_loss: 1.2844 - val_accuracy: 0.4458
Epoch 6/10
9962/9962 [==============================] - 12s 1ms/step - loss: 1.3069 - a
ccuracy: 0.4413 - val_loss: 1.2850 - val_accuracy: 0.4469
Epoch 7/10
9962/9962 [==============================] - 12s 1ms/step - loss: 1.3056 - a
ccuracy: 0.4416 - val_loss: 1.2831 - val_accuracy: 0.4444
Epoch 8/10
9962/9962 [==============================] - 12s 1ms/step - loss: 1.3051 - a
ccuracy: 0.4414 - val_loss: 1.2842 - val_accuracy: 0.4458
Epoch 9/10
9962/9962 [==============================] - 12s 1ms/step - loss: 1.3052 - a
ccuracy: 0.4412 - val_loss: 1.2827 - val_accuracy: 0.4458
Epoch 10/10
9962/9962 [==============================] - 12s 1ms/step - loss: 1.3048 - a
ccuracy: 0.4413 - val_loss: 1.2819 - val_accuracy: 0.4462
2491/2491 [==============================] - 2s 758us/step - loss: 1.2819 -
accuracy: 0.4462
Accuracy: 0.44620808959007263
```

Gaussian Naive Bayes

In [19]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# Assuming 'fcd_df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = final.drop(['type','id'], axis=1)  # Features
y = final['type']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Initialize the MinMaxScaler and fit-transform the training data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)

# Transform the test data using the same scaler
X_test = scaler.transform(X_test)

# Initialize the Gaussian Naive Bayes classifier
clf = GaussianNB()

# Fit the classifier on the training data
clf.fit(X_train, y_train)

# Predict on the test data
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.4064821321019875
```

Best Model Performance on unseen data(New Simulation data)

In [20]:
```
fcd_df_New = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/fcd-out
emission_df_New = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/er
features_New = pd.read_excel("C:/Users/saiko/OneDrive/Desktop/699/Task-7/featu
```

In [21]:
```
# Merging the data using 'id' and 'type'
merged_data_New = pd.merge(fcd_df_New,emission_df_New, on=['time','id','type'
```

In [22]:
```
# Merging the data using 'id' and 'type'
merged_data_New = pd.merge(merged_data_New,features_New, on=['time','id','typ
```

In [23]:
```
final_New = merged_data_New.drop(columns=['slope','eclass','route','waiting',
```

In [24]:
```
from sklearn.preprocessing import LabelEncoder
# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical columns
final_New['type'] = label_encoder.fit_transform(final_New['type'])
final_New['id'] = label_encoder.fit_transform(final_New['id'])
```

In [26]:
```
import pandas as pd

# Assuming df is your DataFrame
# Shuffle the rows using sample() function
shuffled_df = final_New.sample(frac=1, random_state=42)  # frac=1 shuffles the

# Reset the index if needed
shuffled_df.reset_index(drop=True, inplace=True)
```

Random Forest Evaluation

In [27]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Assuming 'df' is your DataFrame containing the data

# Features (X) and target variable (y)
X = shuffled_df.drop(['type','id'], axis=1)  # Features
y = shuffled_df['type']  # Target variable



# Predict on the test data
y_pred = clf_rand.predict(X)

# Calculate accuracy
accuracy = accuracy_score(y, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.5756833535771289
```

In [ ]:
```

```